Chapter 7

1d.

uint64\_t ROR64(uint64\_t u64);

ROR64: //R1,R0 = u64

LSRS R1, R1, 1 // R1 >> 1, C = lsb

ORR R1, R1, R0, LSL 31 // R1[31] = R0[0]

RRX R0, R0 // R0[31] = C

BX LR

3b. uint32\_t BFI(uint32\_t x, uint32\_t y, uint32\_t lsb, uint32\_t len)

BFI: // R0 = x, R1 = y, R2 = lsb, R3 = len

PUSH {R4, LR}

LSL R1, R1, R2 // shift the value left to the start position

LDR R4, =1

LSL R2, R4, R2

LSL R3, R2, R3 // implement BFC so that we can set those bits to 0

SUB R2, R3, R2

BIC R0, R0, R2

EOR R0, R0, R1 // use exclusive or to set those zeroes to the value

POP {R4, PC}

3c. int32\_t SBFX(uint32\_t x, uint32\_t lsb, uint32\_t len)

SBFX: // R0 = x, R1 = lsb, R2 = len

LDR R3, = 32 // save constant 32 in R3

SUB R3, R3, R1 // 32 - (lsb + len) is the # of bit that need to be shifted

SUB R3, R3, R2

LSL R0, R0, R3 // shift the extract from current location to lift

ADD R3, R3, R1 // add R3 with lsb so that we can shift the extract to rightmost

// position

LSR R0, R0, R3 // right shift the extract, during shifting, all bits except the extract

// bit will be set to 0

BX LR

Chapter 8

1.

LSL R1, R0, 6 // R1 = 64 \* R0

ADD R1, R1, R0, LSL 5 // R1 = 64 \* R0 + 32 \* R0

SUB R0, R1, R0, LSL 1 // R0 = R1 - 2 \* R0

3.

28/9 = 256/9 = 28.44

So we should multiply X by 28.

7.

uint32\_t Modulus(int32\_t s32, uint32\_t k)

Give s32 modulus 2k

Modulus: // R0 = s32, R1 = k

LSL R1, R1, 2 // R1 = 2k

SDIV R2, R0, R1 // R2 = quotient

MLS R3, R1, R2, R0 // R3 = remainder

AND R2, R1, R3, ASR 31 // R2 = (remainder < 0) ? divisor: 0

ADD R0, R3, R2 // R0 = modulus

BX LR